大阪フルリージョン 2 周年！
最新情報からメインリージョンとしての活用事例まで教えます！

一歩踏み込んで考える Resilience

アマゾン ウェブ サービス ジャパン合同会社
ソリューションアーキテクト
多田　慎也

自己紹介

名前：多田 慎也
所属：アマゾンウェブサービスジャパン合同会社
　　ソリューションアーキテクト
経歴：インフラエンジニア@SIer
　　-> 西日本担当のソリューションアーキテクト@AWS
好きなAWSサービス：AWS Step Functions, AWS Fargate

本セッションの想定聴講者とゴール

想定聴講者

- AWS の基本的なサービス、概念を知っている方
- システムの信頼性 / 可用性設計に興味がある方
- 東京リージョンと大阪リージョンを使ったマルチリージョン構成に興味がある方

ゴール

- Resilience （回復力）の責任共有モデルを理解できるようになる
- AWS 上でResilience （回復力）を備えたシステムの設計ポイントが理解できるようになる

アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

Resilienceとは

英語で、「回復力」、「抵抗力」、「弾力性」を意味する言葉。

AWS における Resilienceとは
ワークロードが障害に対応し、障害から迅速に復旧する能力

なぜ Resilience が重要なのか？

ビジネスへの影響

- 業務遂行に対する直接的な影響
  メール・チャット、生産システム、ATM、etc
- 企業や組織ブランドへの影響

課題

- これまで以上に Resilience に沿った設計指針が求められている

アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

責任共有モデル for Resilience

お客様

クラウド内のレジリエンスに対する責任

Resilience ‘in’ the cloud

- 重要なインフラストラクチャの継続的なテスト
- ワークロードアーキテクチャー
- 変更管理 / 運用
- Observability / 障害管理
- ネットワーク / クォータ / 制限管理

AWS

クラウドのレジリエンスに対する責任

Resilience ‘of’ the cloud

- ハードウエア / AWS サービス
- コンピュート
- ストレージ
- データベース
- ネットワーキング
- AWSグローバルインフラストラクチャー
- リージョン
- アベイラビリティーゾーン
- エッジロケーション

責任共有モデル for Resilience

お客様

クラウド内のレジリエンスに対する責任

Resilience ‘in’ the cloud

重要なインフラストラクチャの継続的なテスト

ワークロードアーキテクチャー

変更管理 / 運用

Observability / 障害管理

ネットワーク / コンテンツ配信

ハードウェア / AWS サービス

コンピュート

ストレージ

データベース

ネットワーキング

AWS クラウドのレジリエンスに対する責任

Resilience ‘of’ the cloud

AWSグローバルインフラストラクチャー

リージョン

アベイラビリティーゾーン

エッジロケーション

利用者は、Resilience of the Cloudの性質を理解し、
障害への影響をコントロールするため、
適切なResilience in the Cloudを実践することが重要

Resilience of the Cloud

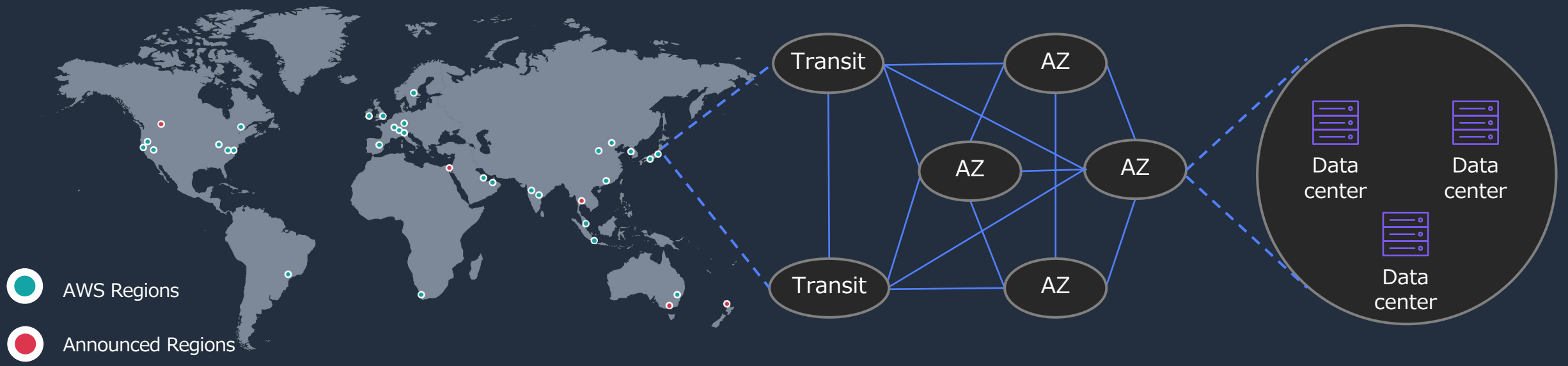
AWS Regions and Availability Zones (AZs)

AWS REGIONS ARE PHYSICAL LOCATIONS AROUND THE WORLD WHERE WE CLUSTER DATA CENTERS

全世界 31 の リージョン

それぞれのリージョンは、複数アベイラビリティーゾーン (AZ) で構成されている

各 AZ は 1 つ以上のデータセンターで構成される



AWS のリージョンは、
他リージョンと完全に分離されるように設計
リソースは指定したリージョンに
結びつけられている

相互に関連する障害を防ぐため,
最大 60 マイル (約 100 km) 離れており、
商用電源、断水、ファイバー分離、地震、火災
、竜巻、洪水などを同時に受けない設計

データセンターは、
それぞれ冗長化した、ネットワーク
、接続、および電源を備えている

Resilience in the Cloud

適切に Resilience in the Cloud を設計するには？

High Availability

設計および運用メカニズムによる一般的な障害に対する耐性

コア サービス、可用性の目標を達成するための設計

Disaster recovery

まれではあるが影響の大きい障害について、目標内に運用に戻る

バックアップ、データ管理、RTO/RPOの管理

Resilience の継続性

CI/CD、Code の改善、運用テスト、Observability / モニタリング、カオスエンジニアリング

可用性について検討する

可用性とは？：ワークロードが使用可能な時間の割合

アプリケーションの可用性における一般的な設計目標 例：

可用性	最大利用不可能時間 (年間)	アプリケーションのカテゴリ
99%	3 日と 15 時間	バッチ処理、データの抽出、転送、ロードのジョブ
99.9%	8 時間 45 分	ナレッジ管理、プロジェクト追跡などの社内ツール
99.95%	4 時間 22 分	オンラインコマース、POS
99.99%	52 分間	動画配信、ブロードキャストのワークロード
99.999%	5 分間	ATM トランザクション、通信のワークロード

https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/reliability-pillar/availability.html

可用性を向上させるには？

$$\text{可用性} = \frac{\text{使用可能な時間}}{\text{全体の時間}} = \frac{\text{MTBF (平均故障間隔)}}{\text{MTBF (平均故障間隔)} + \text{MTTR (平均復旧時間)}}$$

MTBF(平均故障間隔)とMTTR(平均復旧時間)を向上・改善するアプローチが必要

MTBF / MTTR 向上のアプローチ

MTBF (平均故障間隔)を長くする：**耐障害性向上のアプローチ**
- インスタンス/コンポーネントの冗長化(マルチ AZ の活用)
- 過負荷の防止(キャパシティと使用率の確認・負荷制限)
- テストを通じてソフトウェアのバグや欠陥を排除
- 障害範囲の限定化

MTTR (平均復旧時間)を短くする：**障害復旧、運用性向上のアプローチ**
- モニタリング（影響範囲と運用状態の指標を監視）
- 再起動時間が短いサービスの検討（コンテナ・サーバレス等）
- 復旧手順の確立、復旧対応の自動化

適切に Resilience in the Cloud を設計するには？

High Availability

設計および運用メカニズムによる一般的な障害に対する耐性

コア サービス、可用性の目標を達成するための設計

Disaster recovery

まれではあるが影響の大きい障害について、目標内に運用に戻る

バックアップ、データ管理、RTO/RPOの管理

Resilience の継続性

CI/CD、Code の改善、運用テスト、Observability / モニタリング、カオスエンジニアリング

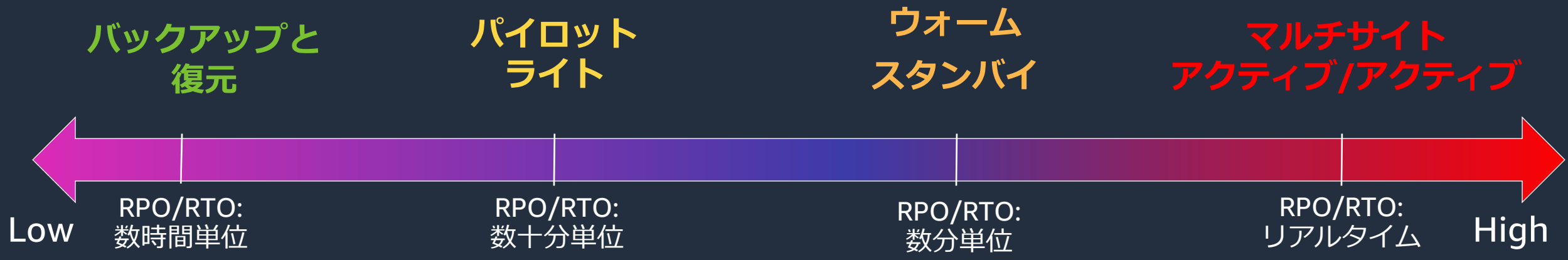
Disaster Recovery (DR) 目標について検討する

- RTO (目標復旧時間) :サービスの中断から
  サービスの復旧までの最大許容遅延時間
- RPO (目標復旧時点) :最後のデータ復旧ポイントからの最大許容時間

どれだけのデータを再作成または、
損失する余裕があるか？

どれくらいのスピードで
復旧しなければならないか？
ダウンタイムにかかるコストは？

AWS におけるディザスタリカバリ (DR) 戦略



- 平常時はバックアップを取得
- RTO に応じて常時サブサイトへバックアップを転送する
- 障害時に必要なデータをサブサイトでリストア
- 費用：$

- 平常時はコアサービスのみに限定したサブサイトを運用 (例 DB のみ常時運用 etc.)
- 障害時に、必要に応じてその他サービスを起動
- 費用：$$

- 平常時から規模を縮小したサブサイトを運用
- 障害時はサブサイトををスケールさせ対応
- 費用：$$$

- 平常時からメインサイト同等構成のサブサイトを運用
- 費用: $$$$

可用性 / DR のニーズ

コンポーネント単位でも検討する

宅配ドライバーの
配送アプリの例

- ドライバーの アサイン機能
- 勤怠管理 機能
- ナビゲーション 機能

システムの重要性	可用性 / DR
高	マルチ AZ / パイロットライト
中	マルチ AZ / バックアップ
低	マルチ AZ /シングルリージョン

システム全体に対して、均一の目標設定をするのでなく、
それぞれのコンポーネントにて、設計目標を**差別化**する

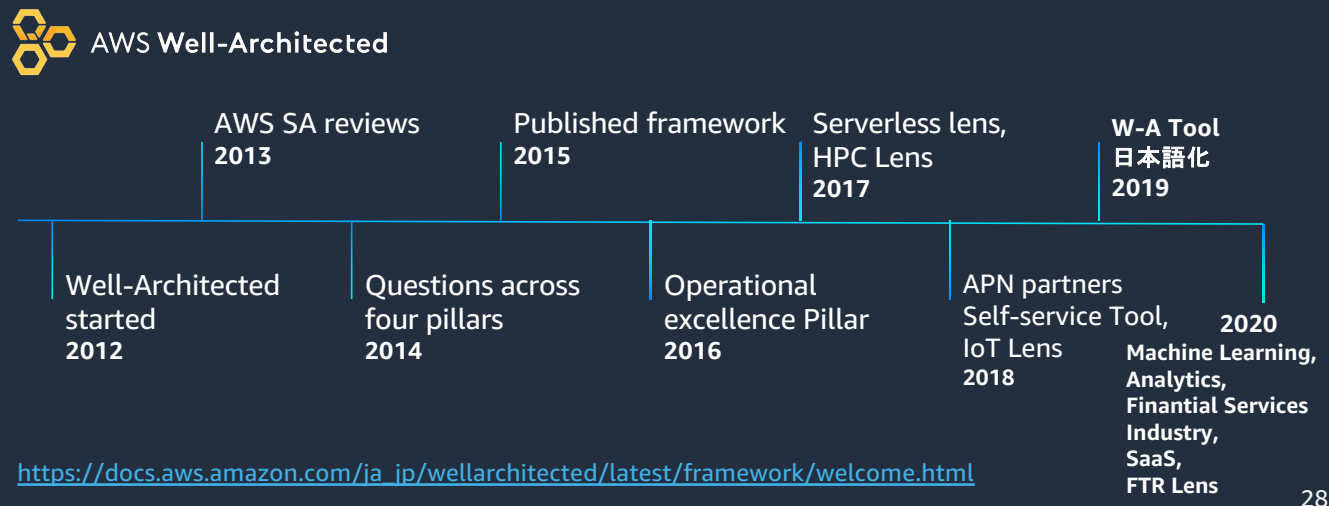
アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

AWS Well-Architected フレームワーク (W-A) とは?

システム設計・運用の”大局的な”考え方と
ベストプラクティス集

- AWS のソリューションアーキテクト (SA)、パートナー様、お客様の 10 年以上にわたる**経験**から作り上げたもの
- AWS と**お客様**と共に、W-A も**常に進化し続ける**



https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/framework/welcome.html

AWS Well-Architected フレームワークの 6 本の柱

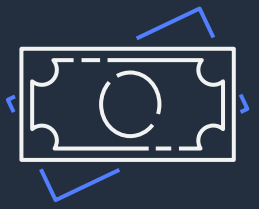
AWS Well-Architected フレームワーク全体の設計原則

運用上の優秀性

セキュリティ

信頼性

パフォーマンス効率



コスト最適化

持続可能性

信頼性における質問

分野	質問のトピック
基盤	REL 1 サービスクォータと制約はどのように管理しますか?
	REL 2 ネットワークトポロジをどのように計画しますか?
ワークロードアーキテクチャー	REL 3 ワークロードサービスアーキテクチャをどのように設計すればよいですか?
	REL 4 障害を防ぐために、分散システムでの操作をどのように設計すればよいですか?
	REL 5 障害を緩和または耐えるために、分散システムの操作をどのように設計しますか?
変更管理	REL 6 ワークロードリソースをモニタリングするにはどうすればよいですか?
	REL 7 需要の変化に適応するようにワークロードを設計するには、どうすればよいですか?
	REL 8 変更はどのように実装するのですか?
障害管理	REL 9 データはどのようにバックアップするのですか?
	REL 10 ワークロードを保護するために、障害分離をどのように使用すればよいですか?
	REL 11 コンポーネントの障害に耐えるようにワークロードを設計するにはどうすればよいですか?
	REL 12 信頼性はどのようにテストすればよいですか?
	REL 13 災害対策 (DR) はどのように計画するのですか?

信頼性における質問

分野	質問のトピック
基盤	REL 1 サービスクォータと制約はどのように管理しますか?
	REL 2 ネットワークトポロジをどのように計画しますか?
ワークロードアーキテクチャー	REL 3 ワークロードサービスアーキテクチャをどのように設計すればよいですか?
	REL 4 障害を防ぐために、分散システムでの操作をどのように設計すればよいですか?
	REL 5 障害を緩和または耐えるために、分散システムの操作をどのように設計しますか?
変更管理	REL 6 ワークロードリソースをモニタリングするにはどうすればよいですか?
	REL 7 需要の変化に適応するようにワークロードを設計するには、どうすればよいですか?
	REL 8 変更はどのように実装するのですか?
障害管理	REL 9 データはどのようにバックアップするのですか?
	REL 10 ワークロードを保護するために、障害分離をどのように使用すればよいですか?
	REL 11 コンポーネントの障害に耐えるようにワークロードを設計するにはどうすればよいですか?
	REL 12 信頼性はどのようにテストすればよいですか?
	REL 13 災害対策 (DR) はどのように計画するのですか?

REL11. コンポーネントの障害に耐えられるようにワークロードを設計する

説明
高い可用性と低い平均復旧時間 (MTTR) の要件を持つワークロードは、回復力を考慮した設計をする必要があります

ベストプラクティス
REL11-BP01 ワークロードのすべてのコンポーネントをモニタリングして障害を検知する
REL11-BP02 正常なリソースにフェイルオーバーする
REL11-BP03 すべてのレイヤーの修復を自動化する
REL11-BP04 復旧中はコントロールプレーンではなくデータプレーンを利用する
REL11-BP05 静的安定性を使用してバイモーダル動作を防止する
REL11-BP06 イベントが可用性に影響する場合に通知を送信する

ベストプラクティスの捉え方

ベストプラクティスにすべて沿っている
必要はありません
重要なのは、ベストプラクティスを理解し
リスクや改善点を認識することです。

アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

“マルチリージョンアーキテクチャで作る必要があります.”

まずはリージョン内の Resilience 向上

- マルチリージョンの前に、リージョン内の Resilience 向上から始めましょう
  - リージョン内で Resilience を高めるエンジニアリングを行っていますか？
  - リージョン内で適切なオペレーションを行っていますか?

マルチリージョン 基本

- 要件を理解する
- データを理解する
- 依存性を理解する
- オペレーションレディネス

マルチリージョン 基本

- 要件を理解する
- データを理解する
- 依存性を理解する
- オペレーションレディネス

ビジネス要件の理解

- 極めて高い回復力の要件が求められますか?
  - 目標を満たさない場合の影響は?
- ワークロードが別のリージョンで実行できることを証明できる必要がありますか?
  - 規制ガイド的アプローチ？
- 要件とコストと複雑さを比較検討する

Tier	Max RTO	Max RPO	Criteria	Cost
Platinum	15 min	5 min	ミッションクリティカル	**$$$**
Gold	15 min – 6 hrs	2 hrs	重要だがミッションクリティカルではない	**$$**
Silver	**6 hrs – few days**	**24 hrs**	**ミッションクリティカルでない**	**$**

正しい理由でマルチリージョンを実行していますか?

可用性の向上

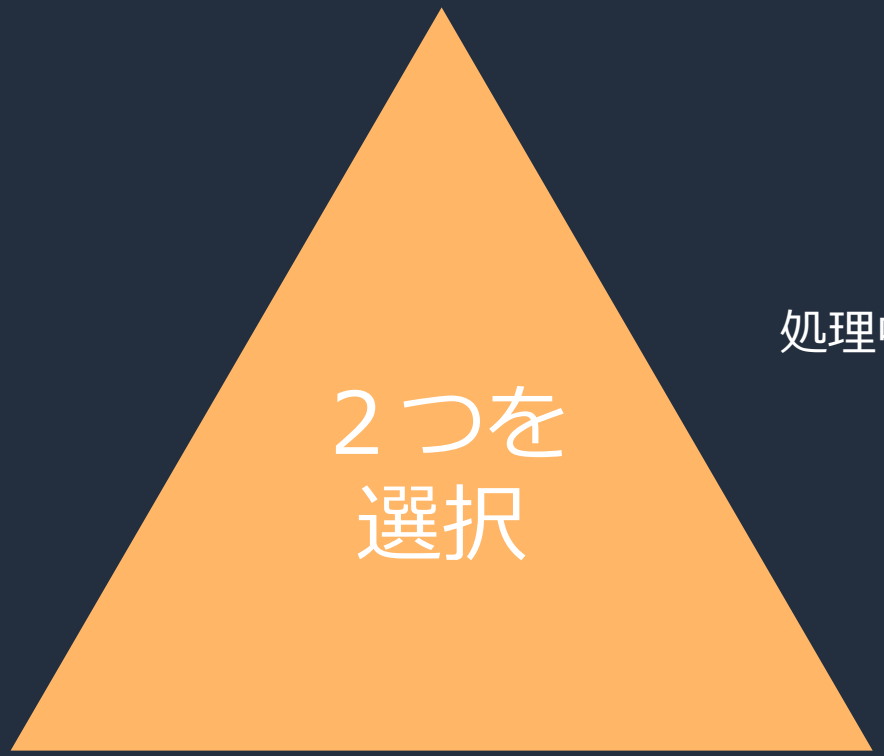
規制上の要求

性能の改善

マルチリージョン 基本

- 要件を理解する
- データを理解する
- 依存性を理解する
- オペレーションレディネス

CAP定理

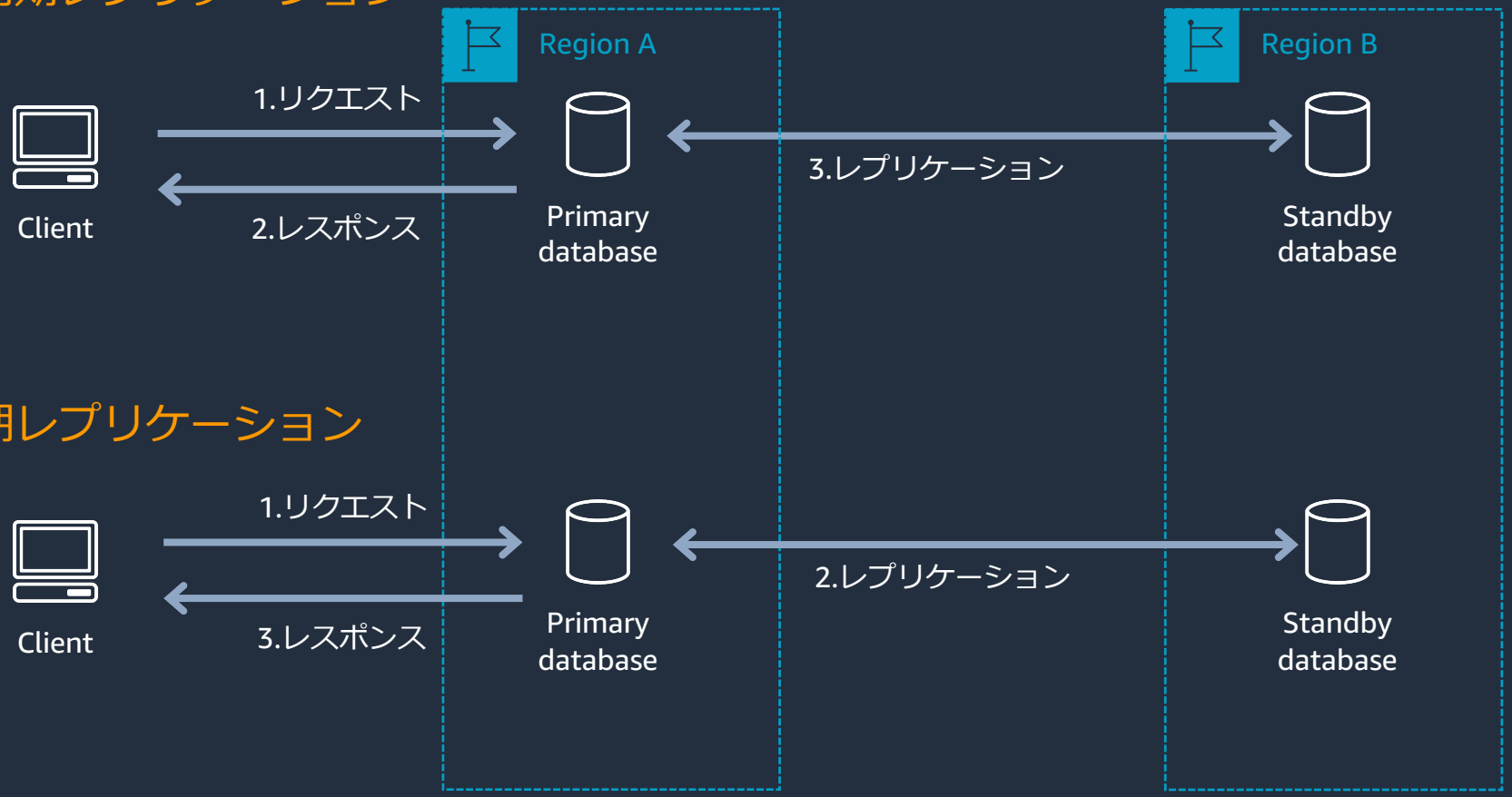


AP
結果整合性,
処理中のトランザクション
のデータロス

CP
パフォーマンス低下,
依存性の追加

データの一貫性の要件

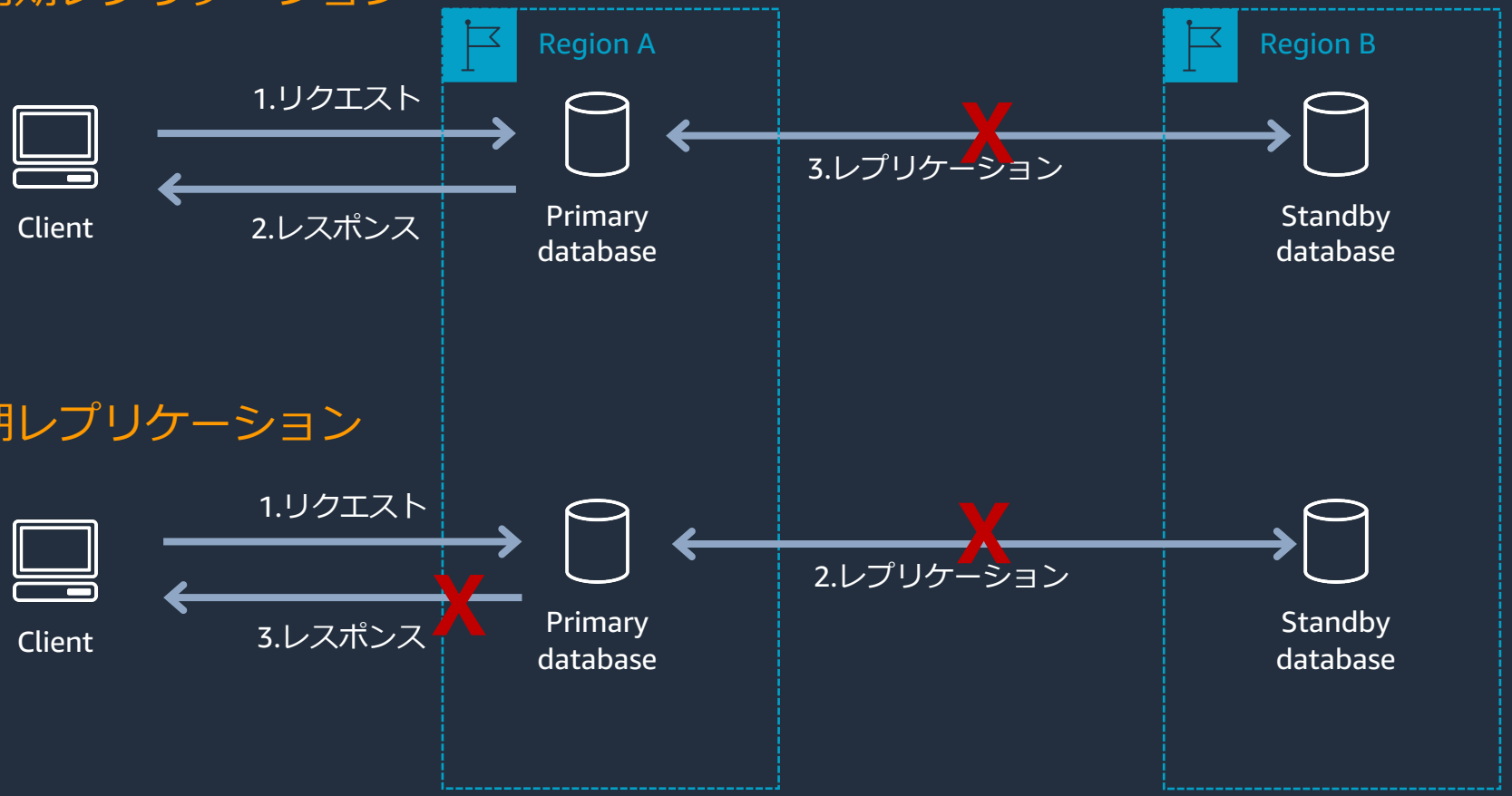
非同期レプリケーション



同期レプリケーション

データの一貫性の要件 (NW分断障害)

非同期レプリケーション



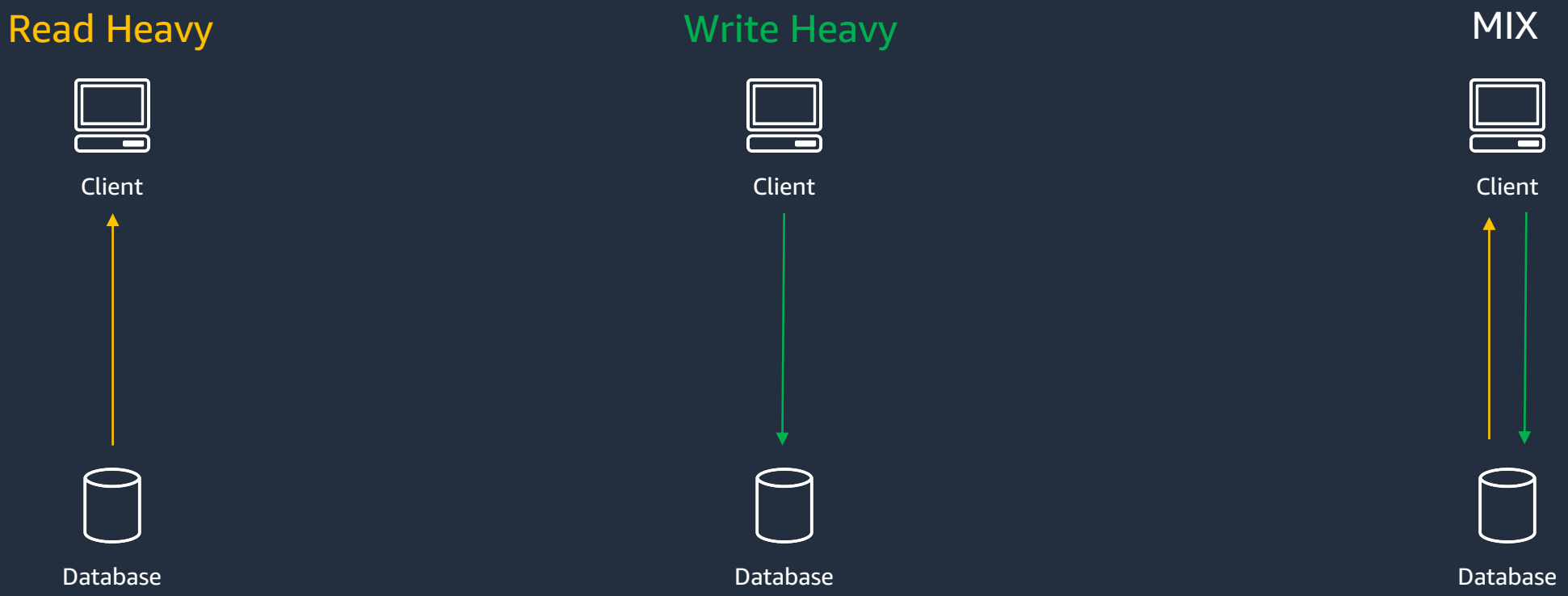
可用性を優先

同期レプリケーション

一貫性を優先

データのアクセスパターン

アクセス特性を理解する



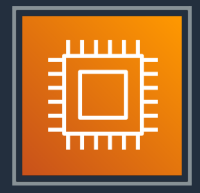
Read
Write

マルチリージョン 基本

- 要件を理解する
- データを理解する
- 依存性を理解する
- オペレーションレディネス

AWS サービス

- 使用するリージョンで利用できるサービスを決定する
- マルチリージョンに役立つ機能は何か特定する


Compute


Storage


Database


Network and Content Delivery

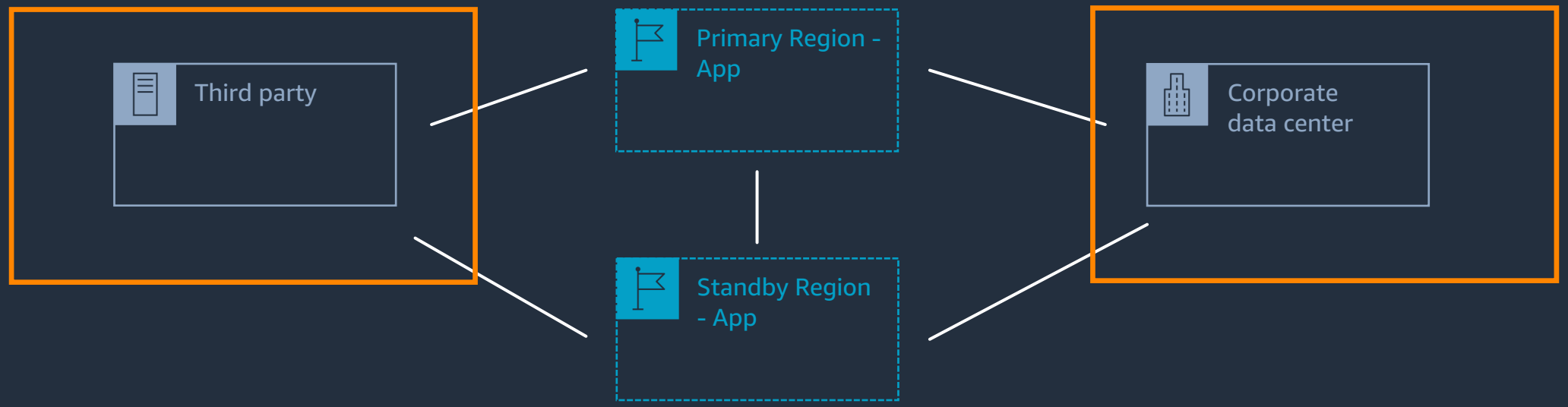

Analytics


Management and Governance


Security

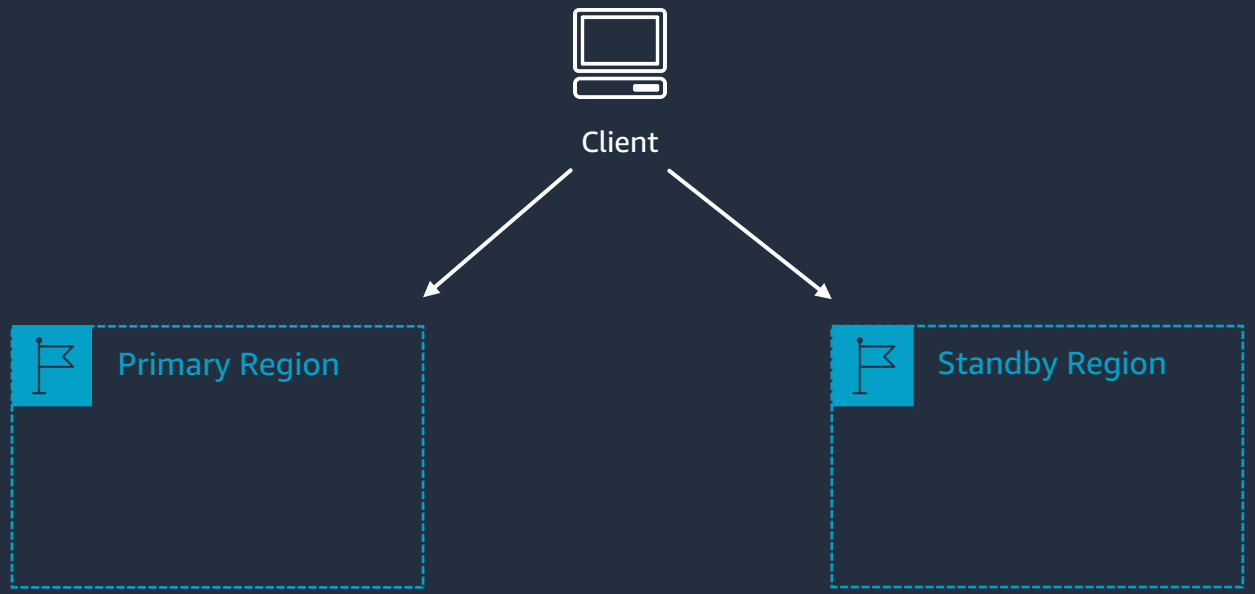
オンプレミスおよびサードパーティへの依存関係

- オンプレミスへの依存関係を特定する
- ワークロード内のサードパーティ ソリューションを特定する



フェイルオーバーメカニズム

- フェイルオーバーのためにプライマリ リージョンに依存しない
- フェイルオーバー メカニズムのすべての依存関係を精査する

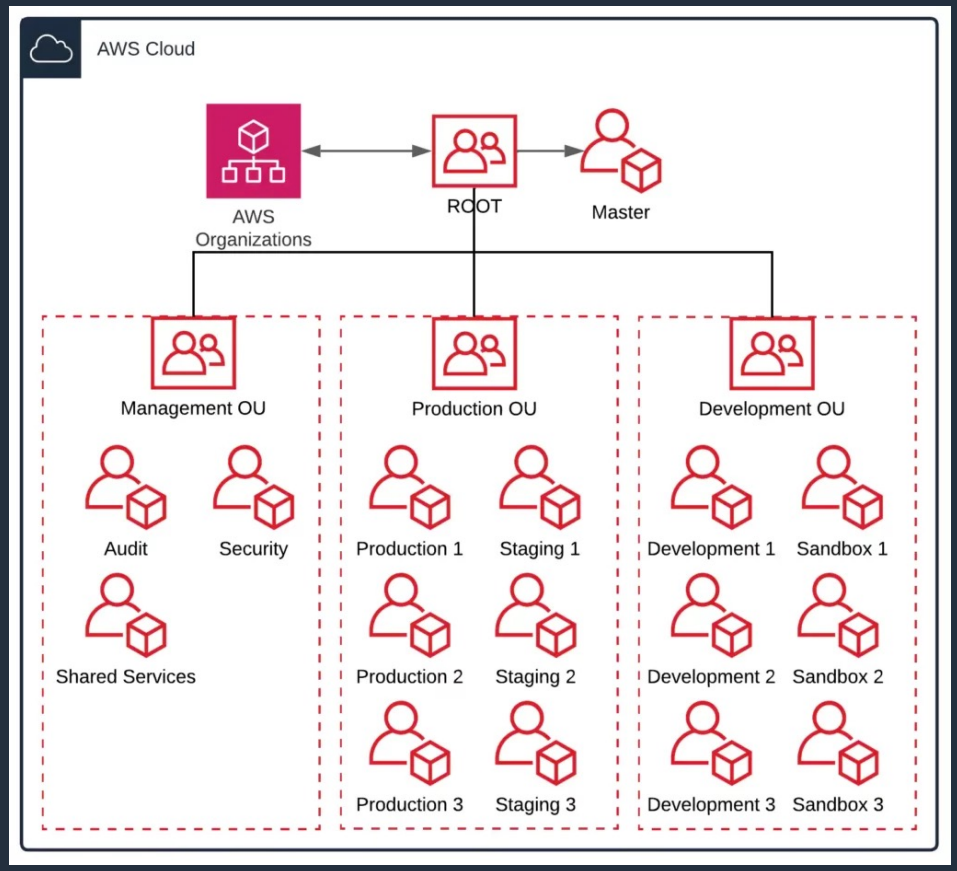


マルチリージョン 基本

- 要件を理解する
- データを理解する
- 依存性を理解する
- オペレーションレディネス

AWS アカウントマネジメント

- AWS サービスクウォータ
- IAM パーミッション
- サービスコントロールポリシー

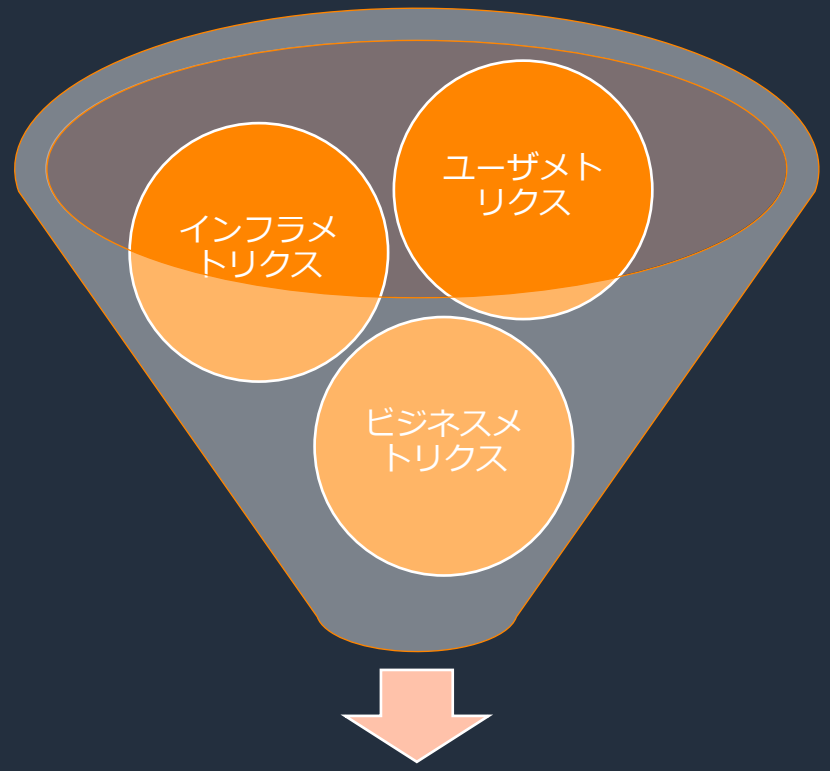


リージョンごとにデプロイする

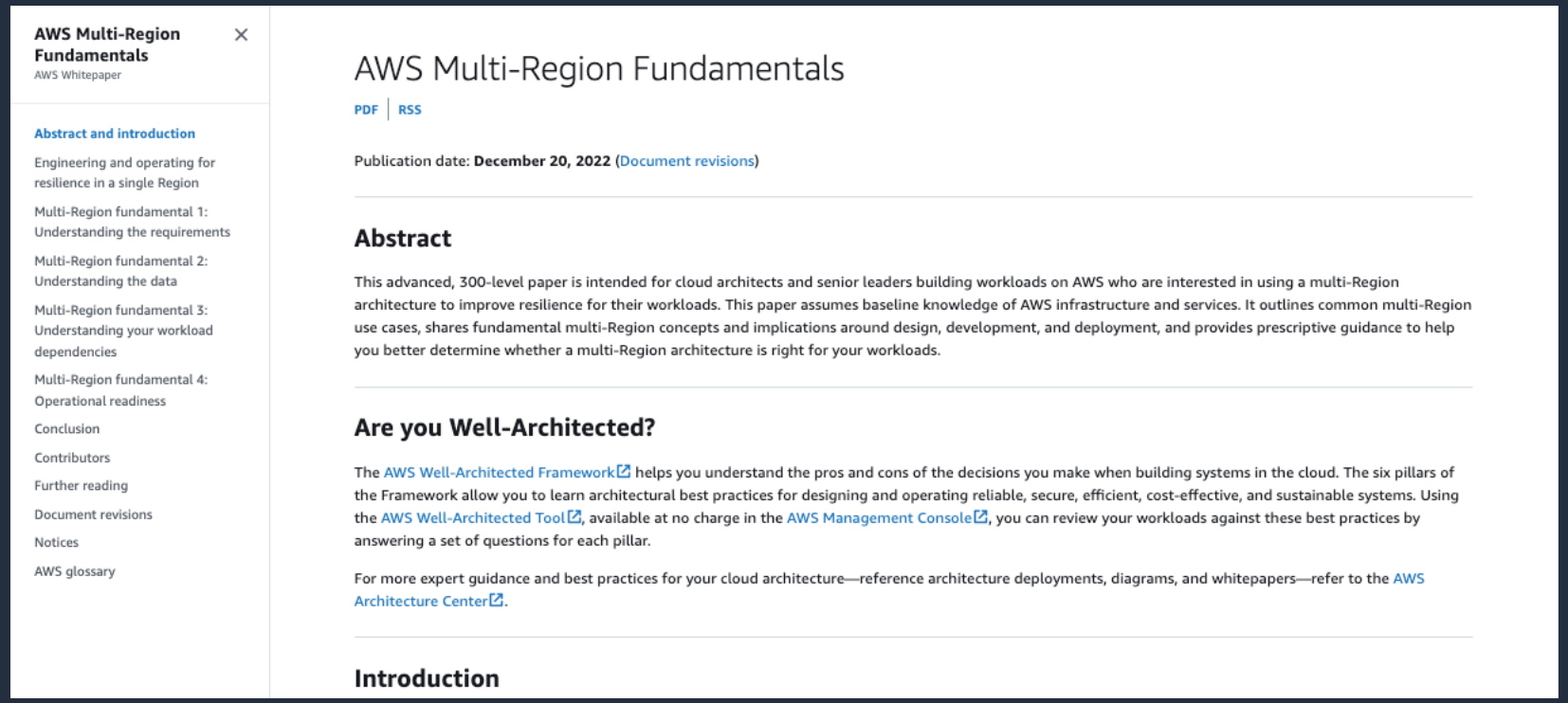
ブルーグリーンデプロイメント または カナリアデプロイメント

プロセスと人

- テストされていない DR 戦略 = DR 戦略が無いのと等しい
- フェイルオーバーの範囲
  - 個々のマイクロサービス?
  - ビジネスの継続性？
- フェイルオーバー メカニズムを定期的にテストする必要があるだけでなく、
  - いつフェールオーバーするかのプロセスを定義する必要がある
- 運用継続計画の一部



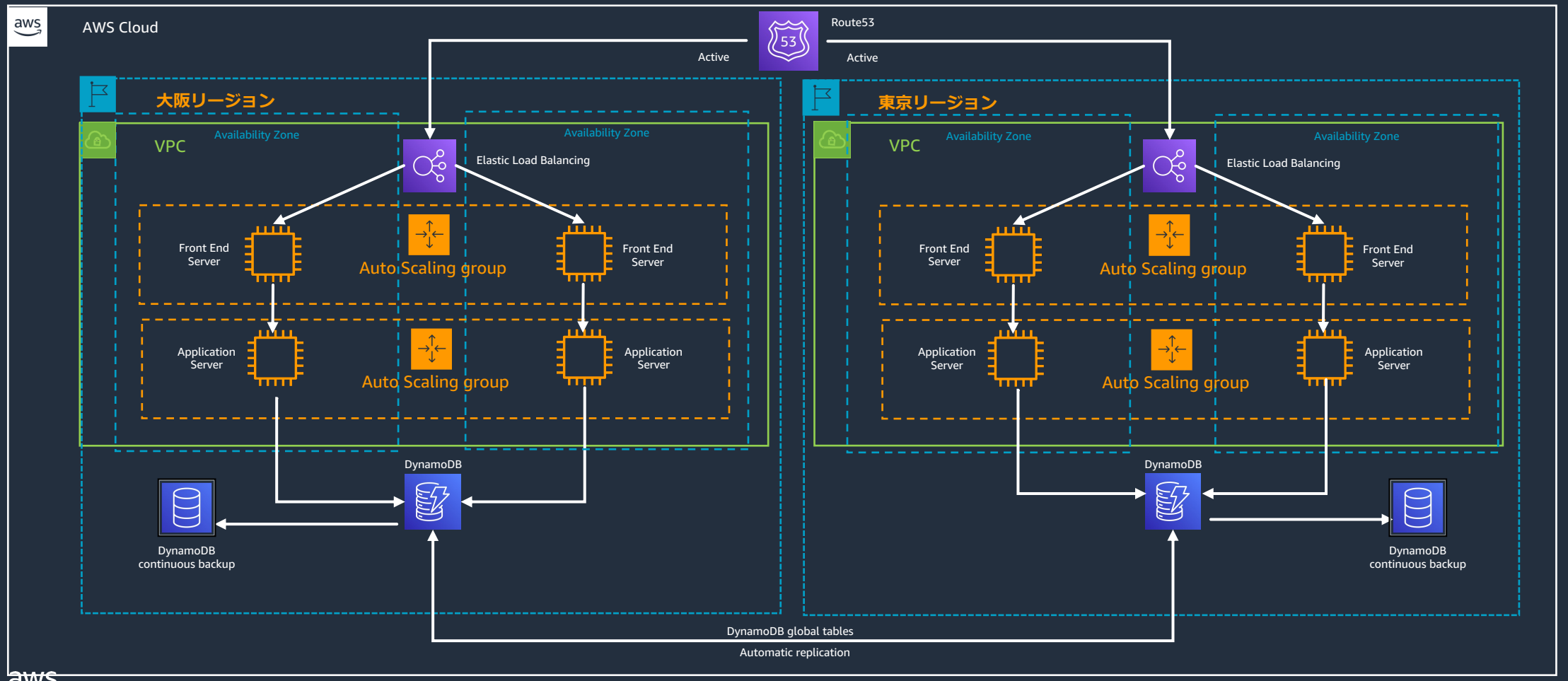
参考： AWS Multi-Region Fundamentals



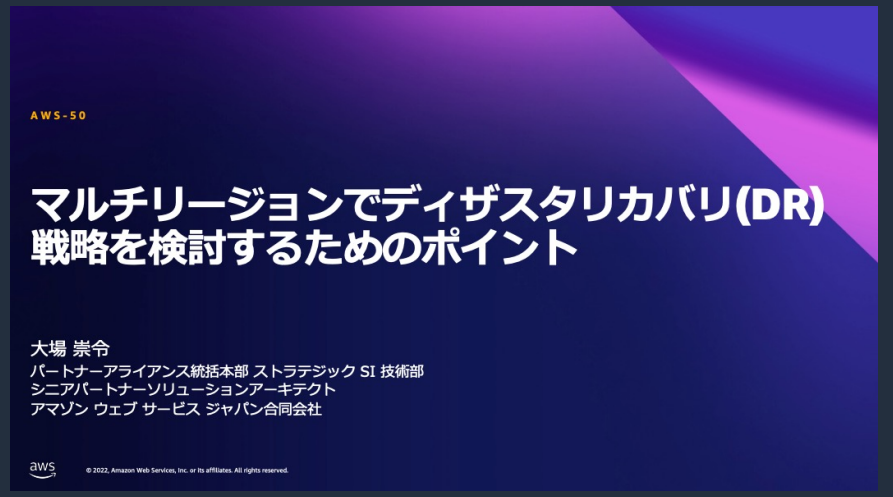
https://docs.aws.amazon.com/whitepapers/latest/aws-multi-region-fundamentals/aws-multi-region-fundamentals.html

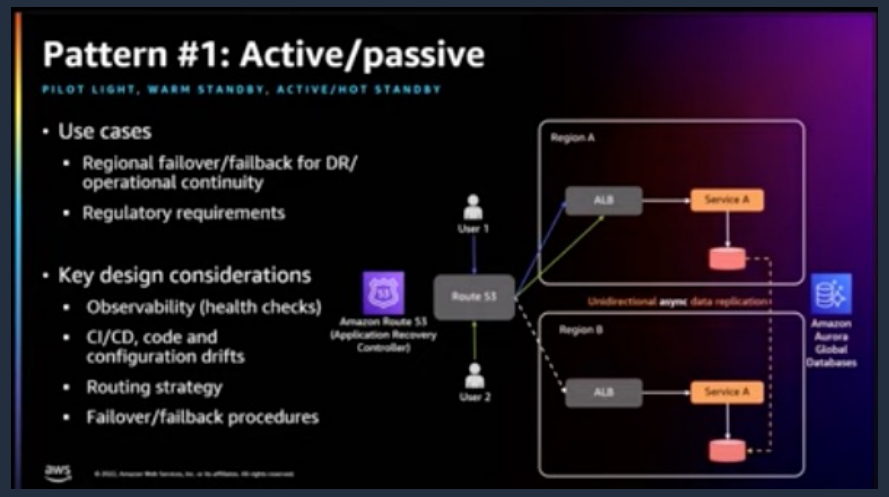
マルチリージョンアーキテクチャの例

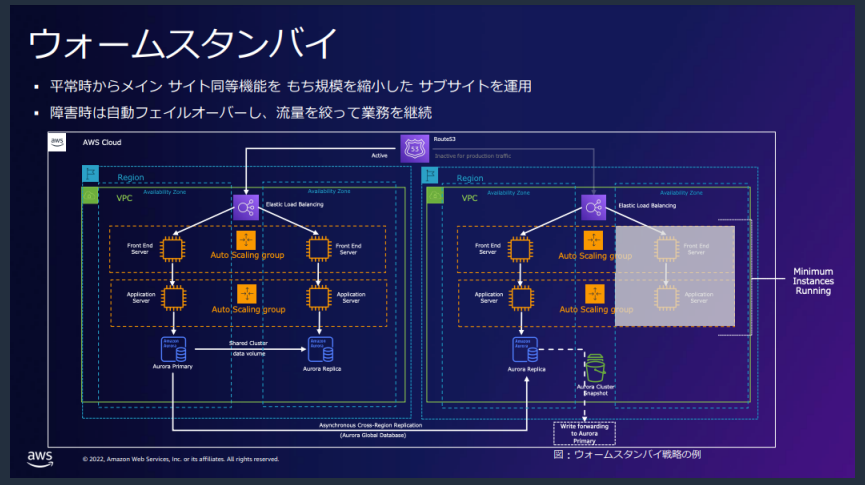
- 平常時からメイン サイト同等構成の サブサイトを運用
- 障害時は 自動フェイルオーバー



参考：マルチリージョン アーキテクチャパターン







https://www.youtube.com/watch?v=ilgpzlE7Hds

https://www.youtube.com/watch?v=bcVK3zN5CIc

アジェンダ

- なぜ Resilience が重要なのか？
- 責任共有モデル for Resilience
- Resilience のあるシステムを作るためのベストプラクティス
- マルチリージョン基本
- まとめ

まとめ

- 責任共有モデル for Resilience
  - お客様の責任範囲：Resilience in the Cloud
  - AWS の責任範囲　：Resilience of the Cloud
- Resilience を備えたシステムの設計ポイント
  - Well-Architected フレームワーク
  - マルチリージョンの基本
    - 要件を理解する
    - データを理解する
    - 依存性を理解する
    - オペレーションレディネス

本セッションのゴール

ゴール

- Resilience （回復力）の責任共有モデルを理解できるようになる

- AWS 上でResilience （回復力）を備えたシステムの設計ポイントが理解できるようになる

Thank you!